The Report Committee for Anish Dhanekula
Certifies that this is the approved version of the following report:

N_2Z – A NFC to ZigBee® Transceiver

APPROVED BY
SUPERVISING COMMITTEE:

Supervisor:

Adnan Aziz

Mark McDermott

N_2Z, A NFC to ZigBee® Transceiver

by

Anish Dhanekula, B.S.E.E.

Report

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

The University of Texas at Austin

December 2012

Acknowledgements

This project would not have been possible without the support of many people. I would like to thank my supervisor, Dr. Adnan Aziz for all his encouragement, support and technical guidance on this project. I would also like to thank Mark McDermott for reading and reviewing the report. I would like to extend special thanks to my colleague Robert Spencer for providing valuable technical support during the course of the project. Finally, I would like to thanks my parents Sailaja & Nagendra Babu, my friend Mehul Jain and my manager Chowdary Yanamadala for their support and help in bringing this project to fruition.

Abstract

N_2Z, A NFC to ZigBee® Transceiver

Anish Dhanekula, M.S.E

The University of Texas at Austin, 2012

Supervisor: Adnan Aziz

ZigBee home automation systems have been around for years. They include smart energy efficient wireless systems that connect devices via mesh networking. It has always been a problem to connect to these ZigBee nodes with existing technologies we use daily. The premise of this project is to provide easy access to these nodes via near field communication, which is a technology that is available on a majority of smart phones today.

The report details the design and implementation of N_2Z, a device that translates data provided by an Android app through a NFC interface to a ZigBee device and vice-versa. We first discuss the requirements and use cases for the device and application. Then the report goes on to discuss the base technologies behind N_2Z, which include the Android platform, Near Field Communication (NFC) & the ZigBee platform. The report then goes on to discuss the design and implementation of the system. Consequently, we discuss the final results of the system with certain use cases and provide some empirical data that gives us insight into system operation. Next we discuss other existing technologies that exist in the market. Finally we conclude the report with a discussion on future work and thoughts on the future of ZigBee.

Table of Contents

List of Tables

List of Figures

Chapter 1: Introduction

A modern home today has many appliances that are controlled by switches - air conditioners, heaters, lighting and more. These devices account for a majority of the energy consumption for a typical household. Many of these household devices have become more efficient through automation. For example, thermostats can be programmed with cooling and heating schedules. They can also adjust flow control by continuously monitoring the ambient room temperature. Other examples include sensors that detect change in room occupancy to automatically turn the light on/off. This kind of home automation can be taken to the next level by connecting these devices through a network. In recent years there has been no shortage of these network enabled control devices for the home environment. Several technologies have been implemented for this purpose, including the ubiquitous protocols such as Bluetooth and Wi-Fi, although over the years ZigBee has become more popular in the home automation market.

ZigBee has its advantages over the more ubiquitous protocols mainly because of its robustness, low cost and low administrative overhead. In spite of these advantages the adoption of ZigBee based home networking systems has been slow. The problem limiting widespread adoption is mainly because of complex and expensive architecture. Most of the existing systems require a computer for the purposes of network management in addition to Wi-Fi + ZigBee modules and expensive ARM processors for translation. These systems can cost hundreds of dollars because of the complexity and administrative overhead involved. However with the revolution in smart phones, a new low cost and low overhead communication channel called NFC is becoming popular.

In this report a NFC based ZigBee translator is proposed and implemented. The system provides a low cost alternative for controlling the back end ZigBee network. With

this device the user can turn any house into smart energy efficient household. The system includes an NFC chip (STM M24LR16E) and a Pic microcontroller to translate the NFC commands to a ZigBee radio (XBEE). Communication through an Android app is also provided as a prototype to enable some basic functionality for powering on/off and simple scheduling tasks. A temperature sensor is also included to provide ambient room temperature data for smart automation of the home thermostat.

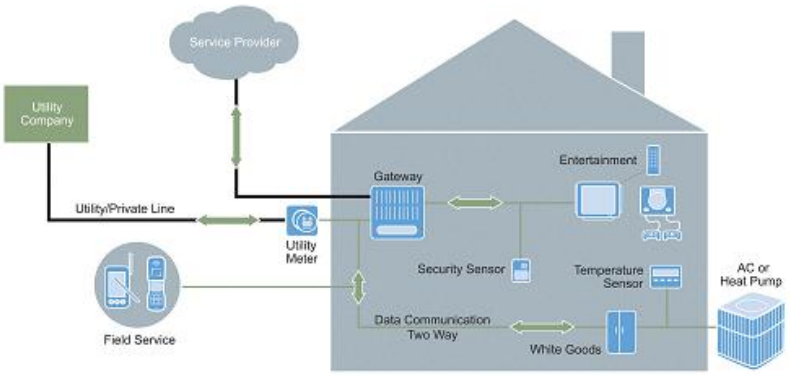

Figure 1: Typical ZigBee home network [1].

USE CASE

An N_2Z device will enable you to interact with your home in a different way. Any power outlet in your home can be controlled from your bedside or the living room or even on your way out. You can use your phone to set a schedule for your sprinklers. When you wake up you can turn on the coffee maker in your kitchen or the heater in the bathroom. With your creativity the possibilities are endless since any power sockets can be controlled and scheduled from any ZigBee installed device in any room. All this can be achieved for a fraction of the cost of existing ZigBee translators [15].

REPORT OUTLINE

This report will first detail the development effort that went into creating the HomeSync Android app. This section primarily talks about the creation of the GUI and provides information regarding the NFC portion of the app. The next section describes the N_2Z hardware platform. This includes channeling the flow of information from the NFC tag to the PIC microcontroller to the ZigBee module. It also describes how the PIC is used to translate commands received via NFC to the ZigBee component of the system. The next chapter specifies the development costs, timeline and testing methodology used for this project. The final chapter summarizes the key contributions and future hardware and software improvements that are required to commercialize the N_2Z system.

Chapter 2: HomeSync, Android App

One of the primary challenges of creating a popular home automation system is to have an interface where the user can gather information about the system in a fast and convenient fashion. There are only two mobile platforms that support NFC – Android and Windows 8. Windows 8 is a relatively new platform and due to this, neither has a sufficient user base nor a good selection of devices for testing. Android has the advantage of being an open source platform and as a result the development process is easier due to vast amount of resources available. For the N_2Z system, Android was the platform of choice for the development of the interface. For more detailed information regarding the technology that makes up the Android platform, please refer to Appendix A. In this section we discuss the design and implementation of the HomeSync Android app that is used to transmit information to the N_2Z device. This chapter is divided into three sections: the GUI, NFC communication and Activity Lifecycle of the app.

GUI

The purpose of the graphical interface is to provide the user a simple straightforward way of viewing the information about the system. This is achieved in Android by using Layouts. There are two ways to define layout elements 1) using XML (Extensible Mark-Up Language) to declare UI elements and 2) creating graphical elements at runtime. Both methods are used in this app for the creation of the GUI. There is an inherent flexibility build into the platform to use the XML and runtime creation simultaneously. In the app the general UI elements are declared in XML and the each element is populated in runtime. This gives you more distinction between the code that generates the UI versus the code that controls the behavior. Basic graphical elements can be described in XML vocabulary, where the attributes are straightforward. Figure 2

shows a description of a On/Off Button where the onClick attribute calls the onSwitchClicked() method of the activity. Using these layout elements, we can build a base view that contains the description of one N_2Z device such as Name, Identifier, etc.

```
<Switch
    android:id="@+id/switch1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:cacheColorHint="@android:color/transparent"
    android:dividerHeight="2dp"
    android:onClick="onSwitchClicked"
    android:gravity="top" />
```

Figure 2: Description of a Switch in XML

Now that we have a base view generated for an N_2Z device, we need to find a way to store a list of these devices and the data fields associated with them. For this project XML was chosen as the data storage format mainly because of its simplicity and the numerous inbuilt resources that Android provides. Android provides several ways to parse XML files. They include Simple API for XML (SAX) method, Document Object Model (DOM) and Pull parsing [8].

The SAX model of parsing an XML file is fast and efficient to implement. It goes through the list in a linear fashion and contents are reported as required. However it is difficult to use for larger databases as the programmer has to keep track of the area of the program that is being processed [8]. The DOM interface is a memory intensive model as it navigates the entire document and creates a tree of objects before you can access the elements. As mentioned earlier, Android provides another method called a pull parser. A pull parser creates an object that sequentially visits the various elements in the XML file. The object can then be used for extracting the tags to determine the start, end and other

data fields of the element. Since the XML file for this system is never going to have large amounts of data, the simplest parser was chosen which was the pull parser. Figure 3 below shows a sample element from the XML database.

```
<item>
 <name>Dining Room</name>
  <uid>0013A2004079D0ED</uid>
 <status>on</status>
 <onschedule>1700</onschedule>
 <offschedule>2300</offschedule>
</item>
<item>
```

Figure 3: Sample XML data element

Now that we have a base view and the XML parser, the onCreate activity now needs to create a number of these views in runtime. Since the content is dynamic and depends on the number of devices in the system, we use a native AdapterView that can populate the layout during runtime. To bring the data from the XML parser into the layout the Adapter subclass is used. Adapter subclass can be viewed as a glue-logic between the data and the AdapterView layout. Figure 4 below shows the final results of this approach.

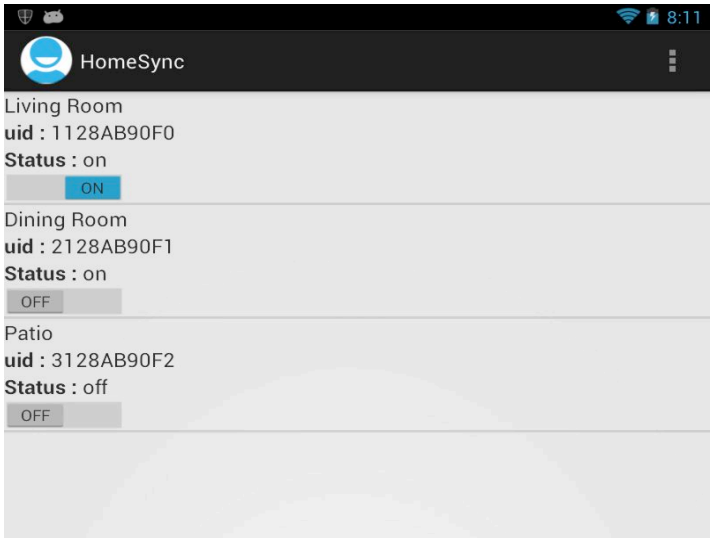

Figure 4: HomeSync GUI

NFC

The primary function of the app is to transfer the information to and from the N_2Z device. Android is backing NFC heavily so as a result many phones have this feature. The technology is ideal for sending small payloads of information between the phone and N_2Z device without much overhead. Android has a set of predefined APIs that assist in accessing this data from the NFC tag (N_2Z Device). A majority of the APIs are based around a format called NDEF (NFC Data Exchange Format) that was defined by the NFC Forum. NDEF is a lightweight, binary message format that can contain multiple application specific payloads in a single message [9]. However the NFC device (M24LR16E-R) that we have selected for this project does not support the NDEF format. For communicating with such devices we are required to define our own protocol stack. For the purposes of making the project modular we define a class `stm24lr16e` that represents the various functions that are available in the device such as Inventory command, Write & Read Memory commands. The above class is analogous to a driver,

so if a need ever arises to use a different part we can create a different driver for the device without changing much code in the central state machine.

Android-powered devices are always looking for NFC tags within range when the phone is unlocked. Once the tag is in range Android provides a tag dispatch system that analyzes scanned NFC tags and tries to find applications that it thinks are interested in the data. The mechanism of this dispatch system is illustrated in Figure 5. This system works on a priority basis where it tries to start an activity with the intent ACTION_NDEF_DISCOVERED being used. If it's not an NDEF formatted tag that is true in our case it tries to start an activity with the ACTION_TECH_DISCOVERED or TAG_DISCOVERED intent [2].

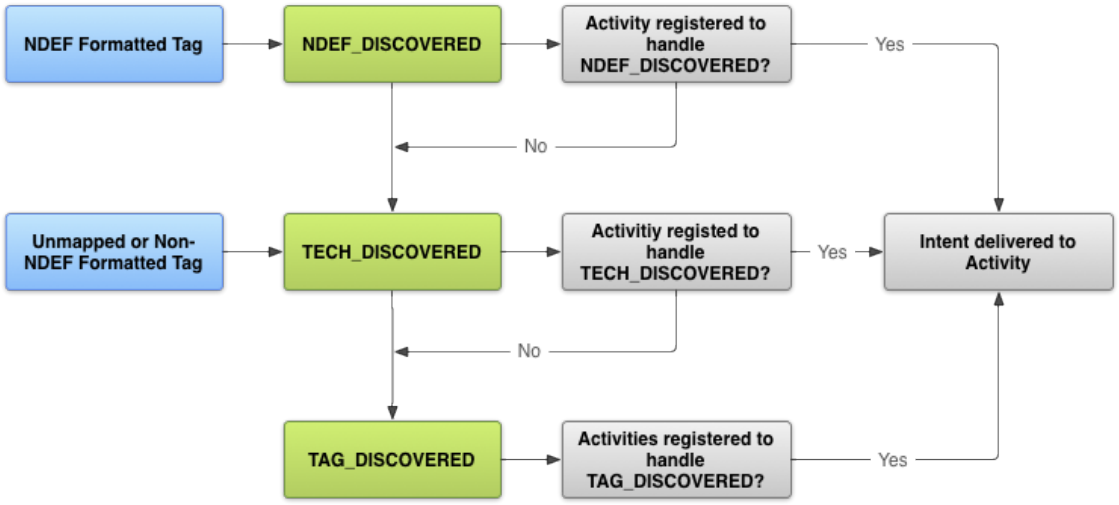

Figure 5: Tag Dispatch System [2].

In order to use the NFC hardware on the device the NFC permissions needs to be defined in the AndroidManifest.xml. The `<uses-permission>` and `<uses-feature>` elements need to be populated appropriately in order for the app to function

as intended. Once the permissions are set appropriately Android can pass on the filtered NFC intents to the app, in our case we use the ACTION_TECH_DISCOVERED intent. This intent is for tags that do not support the NDEF format but support a specific set of technologies that can be found in the Android documentation. The underlying purpose of this intent is to run an inventory command in the listed technologies and wait for a response from the tag. This response can be read by calling `getTechList` to find out the technologies the tag supports. The technology we use in the tag corresponds to NfcA in the tech-list which supports ISO 14443-3A.

HOMESYNC ACTIVITY LIFECYCLE

Figure 6 below illustrates the activity lifecycle of the HomeSync app in relation to the life cycle pyramid shown in Chapter 2. A high-level view of the `N2Z` class is also shown below.

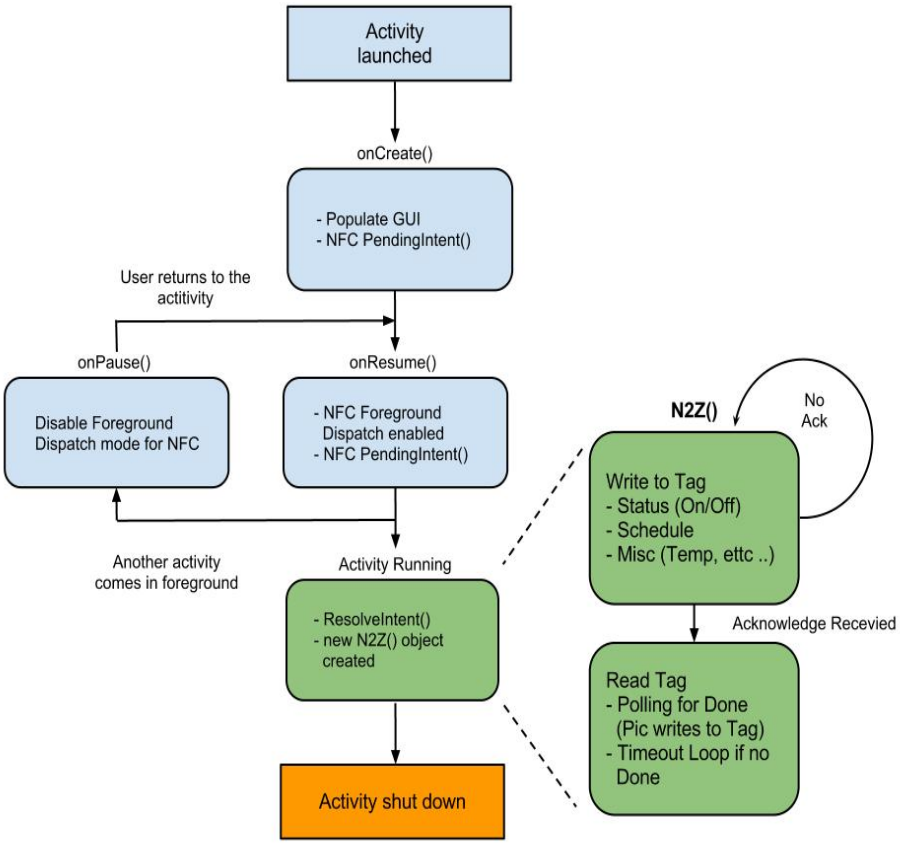

Figure 6: HomeSync app activity lifecycle

Chapter 3: N_2Z, Hardware Platform

In this section we discuss the hardware behind the N_2Z system. The device consists of three distinct parts, the NFC chip, PIC Microcontroller and the ZigBee Device (XBEE).

M24LR16E-R

Performing the NFC duties on the device side is the M24LR16E from STMicro. It is a dual interface device with a 16K bits EEPROM [11]. The memory can be accessed with both interfaces the NFC and the other interface being I2C. The chip also includes an energy-harvesting feature that can be used to power another chip for limited amounts of time. The device behaves as slave in the I2C protocol and observes standard protocol where Read and Write operations to the memory are initiated by a Start condition written by the bus master (PIC1845K20). The contactless interface is complaint with the ISO15693/ISO18000-3 standard discussed in more detail in Appendix B. Incoming data is received as an ASK wave which can be 10% or 100% modulated. Outgoing data is generated using Manchester coding on frequencies 423 kHz & 484 kHz. There are 2 data speed options available the slow mode being 6.6Kbit/s and the fast mode being 53Kbits/s. The default communication mode for this part is 100% modulation, single carrier and fast mode (53Kbits/s) [4]. The sequence of events between the vicinity coupling device (VCD, phone in our case) and the vicinity integrated circuit card (VICC, M24LR16E-R) happens as follows. First, the RF field from the VCD activates the tag. The VCD then transmits a command, which is then followed by as response from the tag. This technique is referred to as Reader talk First (RTF) [11].

Power is derived from the RF energy field through the antenna connected to the AC0 & AC1 pins. No external power is required to power the tag. To ensure efficient

power transfer from the VCC to the tag a loop antenna is used. Figure 7 shows the loop antenna board that was used for the project. Loop antennas are ideal for this application because of their smaller sizes and directivity [10]. The resonant frequency of a loop antenna is shown in the equation below [11]. The capacitance can be obtained from the data sheet and is found to be 27.5 pF and f is known to be 13.56 MHz.

$$f = \frac{1}{2\pi\sqrt{L \,.\, C}}$$

$$\boldsymbol{L = \frac{1}{(2\pi f)^2 . C} = 5uh}$$

The inductance of the antenna is calculated to be 5uh. From this figure the number of turns required for a conductor loop can be calculated using the equation below. This is an approximation based on the assumption that d, the diameter of the conductor coil, is very small compared to D, the diameter of the conductor loop. N is calculated to be around 9 turns. A copper wire was used to build the antenna with 9 turns and a diameter of 1.5cm. The communication with this antenna was very intermittent because of the hand built nature of the structure.

$$L = N^2 \mu R . ln \frac{D}{d}$$

Figure 7: ANT-M24LR-A Antenna Board for NFC Tag

PIC MICROCONTROLLER (PIC1845K20)

The microcontroller used is a PIC 1845K20, which was chosen mainly because of its ubiquity and low cost of development. The development kits are inexpensive and the MPLAB IDE is available as a free download. The MCU also has low power consumption modes that make it ideal for our application. The highest frequency that can be achieved from the internal source is 16MHz. However there is a frequency multiplier mode available that can be enabled to achieve a 4x speed improvement to 64Mhz [12]. It should be noted that the internal oscillator is not the most accurate source; it would be preferable to use an external crystal for better accuracy. The IDLEN bit is turned on which ensures that the clock goes into idle mode when the sleep instruction is executed. Each instruction on the PIC is equivalent to 4 clock cycles [12]. The microcontroller also includes UART and I2C functionalities that are used to communicate with the XBEE module and the NFC tag respectively.

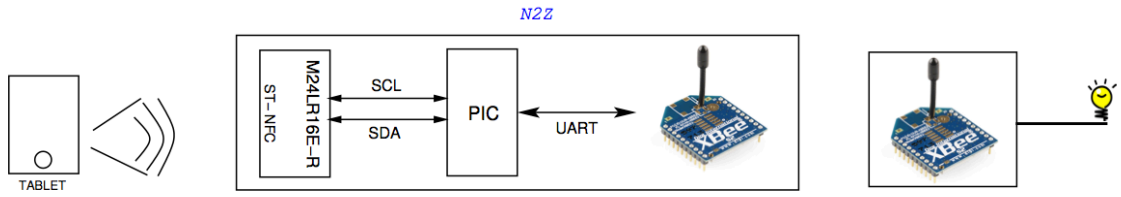

Figure 8: Remote Command Request Frame format

The PIC is sandwiched between the NFC tag and the XBEE where it acts as a translator, as shown in Figure 8. The NFC tag communicates with the PIC using I2C protocol. The pins used are RC3 (SDA) and RC4 (SCL) and they are tied to a pull up resistors so the external line is not floating when the PIC is not driving a value. High-Level functions were available for communicating via I2C but I was unable to read the correct data back from the device. So instead I implemented a few subroutines that manually control SDA & SCL to mimic I2C Start, Stop, Write & Read bytes commands.

At start-up the micro initializes all the necessary settings and ports and immediately goes to sleep. Once in sleep mode it waits for an interrupt to wake up and process the I2C commands. The NFC tag generates the interrupt under the presence of a VCD (Phone). This is made possible by the power-harvesting feature of the tag. Once the interrupt is generated the micro cannot communicate immediately with the tag via I2C. This is because the tag can only communicate with only one interface at a time, either NFC or I2C. In order to account for this loss of communication wait states were included before an I2C command can be sent.

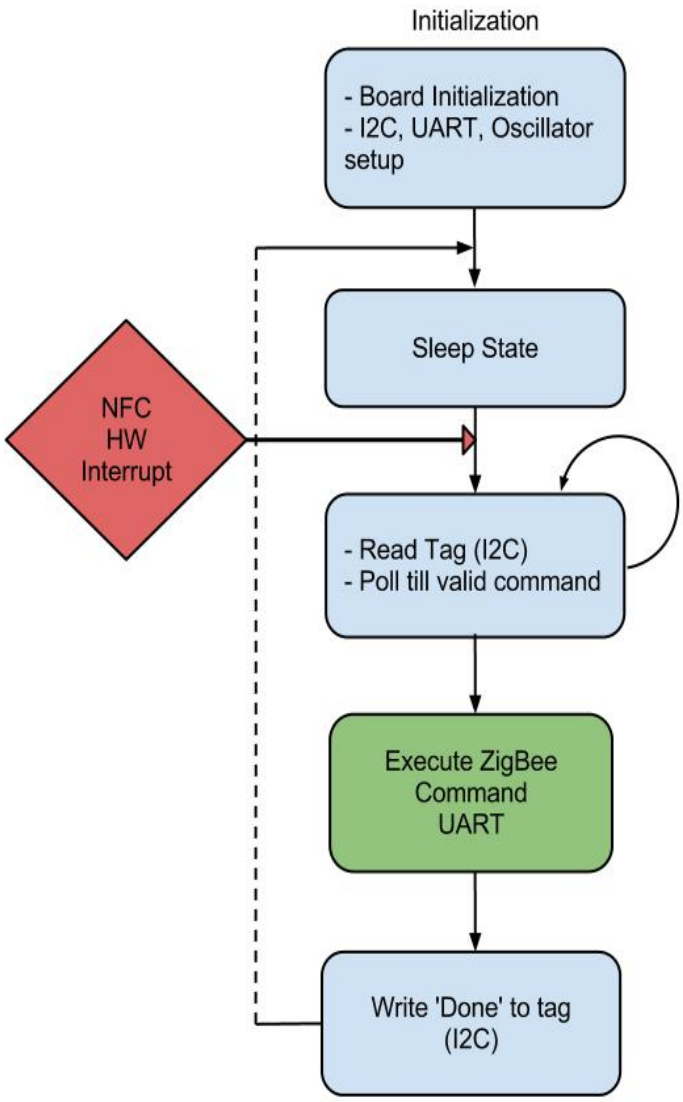

Figure 9: PIC State Machine diagram

Since NFC is based on proximity, sometimes the communication is not as robust as hoped. One of the main reasons for this intermittency is because of the placement of

the tag near the phone has to be very accurate and this is not always possible since the position of the antenna inside the phone is not known. In order to make sure the communication went through we incorporate some basic handshaking mechanisms. When the phone communicated with a tag it writes a particular value to the EEPROM location in the tag. When the PIC reads that location and does not find the expected value the state machine is stalled because it perceives it as an invalid command. These kind of safeguards need to be built to make sure we have a robust system where wrong information does not get transmitted. Figure 9 above illustrates the PIC state machine.

Once we read the command from the tag the PIC needs to translate that command and send it to the XBEE module. The communication between the PIC and the XBEE is done via the USART. The UART communication data rate can be set on the XBEE and is set to 115200bps. But due to the inaccuracy of the internal PLL present on the PIC the data rate achieved on the micro is 114.29K. The baud rate for the UART on the PIC is calculated with the equation below where SPBRGH:SPBRG register pair needs to be set to achieve the desired baud rate and Fosc = 64MHz [12]. The SPBRGH and SPBRG registers make up for the high and low byte of the 16-bit value that determines the period of the free running baud rate timer [12].

$$Desired\ Buad\ Rate = \frac{Fosc}{16([SPBRGH:SPBRG]+1)}$$

$$SPBRGH:SPBRG = \frac{\frac{Fosc}{Desired\ Baud\ Rate}}{64} - 1 = 8$$

$$Calculate\ Baud\ Rate = \frac{64000000}{64(8+1)} = 111.11kbps$$

$$Error = \frac{115200 - 111.11}{115200} = 3.55\ \%$$

A 3.55% error in frequency is considered a significant discrepancy in the data transfer rates. As a result of this the communication can be slightly unpredictable. To mask such inefficiencies each ZigBee command is run five times in a loop to make sure the message is transmitted as intended. The XBEE module is then put into sleep mode for conserving power. In the appendix it is mentioned that a network coordinator is usually not allowed to be in a sleep state. But for our application model the coordinator initiates all the communication and can therefore go to a power save mode once the transmission is complete.

XBEE

The ZigBee device used for this project is a Digi Xbee module with a built in antenna. The module can communicate with the PIC through a UART. Prior to communicating with the PIC, the XBEE needs to be setup as a Coordinator for the purpose of the application. This can be done with the X-CTU software provided by Digi through the USB port on the computer using an USB-Serial adapter. Once the latest coordinator firmware is downloaded and installed the XBEE needs to be set in API mode and the Baud Rate needs to be set to 115.2Kbps in order to communicate with the PIC.

The XBEE handles two communication formats, AT mode and API mode [13]. The AT commands set are easy to use and human readable. It has two modes of operation command mode and transparent mode. Transparent mode is the default state of the XBEE in AT mode. This mode is used for sending data through the XBEE to an endpoint/router device. When data is received it is again transmitted straight through the XBEE into the serial port. When you want to send commands directly to the XBEE instead of a remote XBEE the command mode is available. In both these modes the communication is very simple and can easily be decoded by a human. The AT mode is perfect for setting up a pair of XBEE's initially to prove that they can talk to each other.

Start Frame	64-bit Destination ID	16-bit Network Address	Remote command	Command Parameter	Checksum

Figure 10: Remote Command Request Frame format

However for more data intensive and structured operations the API mode is available. When the XBEE is loaded with the API firmware a more structured command set is available to communicate with the microcontroller. An example of an API command is a ZigBee Transmit Request command. This command is used to transmit data from the coordinator to the router/end device [14]. Figure 10 above shows the structure of the Remote Command Request. The start frame consists of a start delimiter and length of the rest of the packet. The 64-bit address is the address of the Xbee module that you want to control. The remote command can be any of the AT commands. One such command is the ATD0 command which configures the I/O pin as an Analog or Digital pin [14]. The output of the mode of the digital pin is also controlled through this command; this is how the remote control of the XBEE module is achieved. A checksum is issued at the end to ensure data integrity.

Chapter 4: Summary of Results

In the report above, the hardware and software engineering required to build a NFC to ZigBee translator has been demonstrated. Also a communication medium was achieved between a tablet and the N_2Z system in the form of an Android application.

BILL OF MATERIALS

Table 5.1 below details the cost of building a N_2Z device and the costs for testing the ZigBee system. Some assumptions were made at the beginning of the project regarding the costs. One assumes that the user already has a NFC enabled smartphone or tablet and the app would be downloaded from the Google Play Market for free. For testing the N_2Z to ZigBee transmission a PowerSwitch tail was paired with an XBEE to power on/off electrical sockets. The PowerSwitch tail is a 120V AC adapter that can be controlled with a DC input. The DC input is provided by the XBEE (Figure 12). The XBEE for this purpose was set as an end point device using the X-CTU software provided by Digi. The NFC enabled tablet that was used was a $200 Nexus 7 running Android 4.1 (Jelly Bean). The total cost of the N_2Z device, shown below in Figure 11, was calculated to be under $70, but this includes the cost of the antenna from STMicro. Technically this antenna could be replaced with a well-assembled copper wire, which is of negligible cost and would bring the device cost below 30$. A video of the N_2Z system in action can be found in Appendix D.

N_2Z Device		N_2Z Testing	
Part	Cost ($)	Part	Cost ($)
XBEE from Digi PIC 18f45k20 STM24LR16E-R ANT-M24LR-A from STMicro (Antenna Board)	24.95 2.63 1.85 36.77	XBEE x2 PowerSwitchTail from Sparkfun Nexus 7 Tablet	49.90 25.99 200.00
Total	$66.20	Total	$275.89

Table 1: Development cost of N_2Z system and testing hardware

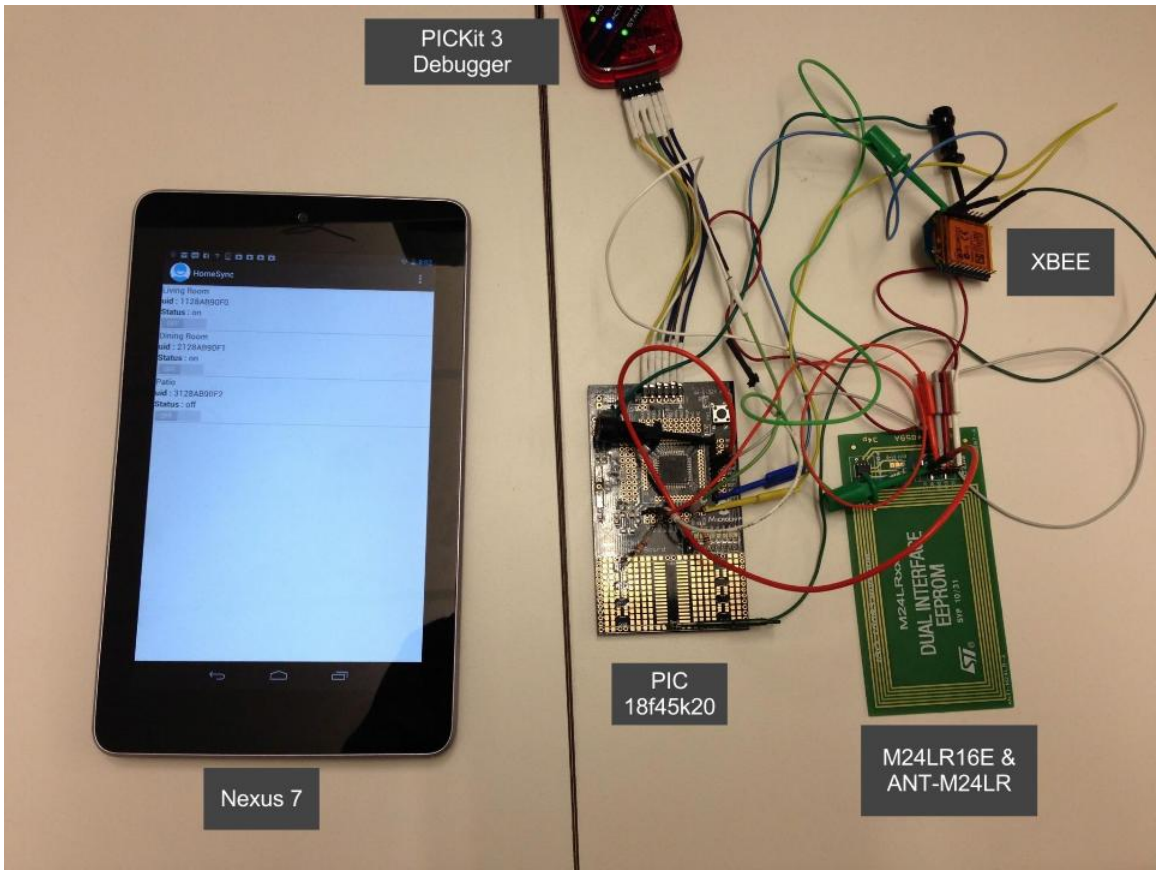

Figure 11: The N_2Z System

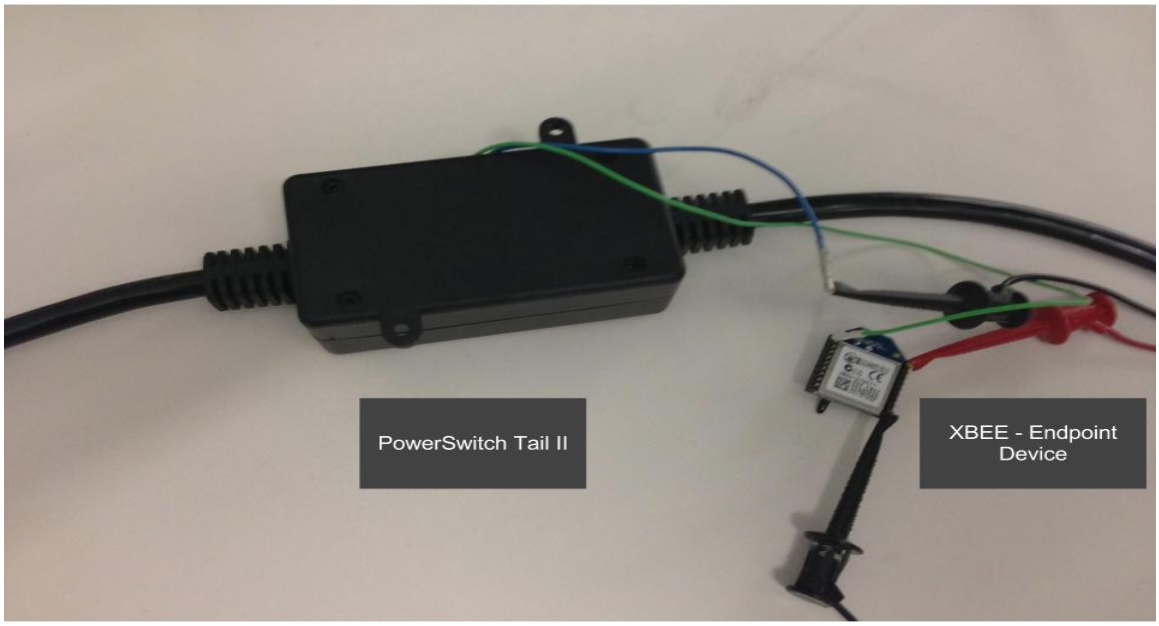

Figure 12: ZigBee power adapter

DEVELOPMENT SCHEDULE AND CHALLENGES

The development of the project took a little more than two months with 50% effort during that time period. Table 5.2 below shows a breakdown of the project duration for each major section of the project. The project was completed with a short accelerated schedule because of missteps in planning phase of the project. The development of the Android app took 4 weeks with equal time spent between building the framework and enabling the NFC communication. This included learning Java and learning the basics of building and debugging an app on Android using the SDK and Eclipse development environment. The most challenging part of the Android development was learning Java. Previously, I have done a few projects in C before but object oriented programming is a new topic to me. Another challenge was the use of Intents to access the NFC adapter. It took 100 percent of my time for a week to figure out how to parse intents. Once that was

figured out it did took another two weeks to build the `stm24lr16e` class, which includes all the tools necessary for communicating with the tag. This accelerated schedule would not be possible if it was not for the thorough documentation provided on developer.android.com.

Building the N_2Z system took about the same time as the app development effort. The biggest challenge on the device side was the NFC integration again. Enabling the power harvesting to work adequately was a difficult task. This was due to the bad quality of the antenna and the strength of the field from the tablet being used. After a fair amount of tries I abandoned the effort to build an antenna and purchased a board provided by STMicro with a built in antenna, which I was not aware of at the beginning of the project. The PIC development was the bulk of the project and it included the effort to communicate with the NFC device as well as the ZigBee device. It also included the work for creating the state machine discussed in Chapter 4.

Section	Duration (weeks)
Android App Development	**5**
- Basic Framework	2
- NFC Communication	3
N_2Z **Development**	**5**
- NFC integration	2
- ZigBee integration	1
- Microcontroller development	3
Total Duration	10 weeks

Table 2: N_2Z & HomeSync development timeline

Chapter 5: Conclusion and Future Work

A NFC to ZigBee translator was proposed in this report. A proof of concept was presented and demonstrated during the course of project. A base N_2Z device was found to cost under $30 a wireless switch to accompany this can be made for another $30 or can be bought from other manufacturers for $50. Existing solutions with similar features cost around $300 for the translator alone and based on Wi-Fi [15]. So for the cost of one of these devices a user can actually hook up an entire house with a N_2Z powered system. In order to reach this goal there are several improvements that have be made to the system. This section will describe the various enhancements that can be made to the project both in hardware and software for future development.

SOFTWARE ENHANCEMENTS

While the GUI developed in this project did a good job of presenting the system data, more work could be done so we can add more multiple devices. Also more functionality could be added to the current power options and scheduling, like reading the power consumed by each source. The app could also be made to serve notification when a current schedule gets activated to serve as a reminder. In present form there is no security layer to the device, but one can be added with simple asymmetric authentication protocol to prevent other NFC devices from talking to the system. This adds to security already inherent to the system because of the proximity needed to access the technology. Also the app was developed only with the Nexus 7 running Android 4.1 in mind, with a little bit of work the app could be made available to a more diverse set of phones/tablets.

HARDWARE DEVELOPMENT

The number of components on the hardware side is relative small; because of this we can develop a small PCB that can be enclosed in an extremely mobile package. Also

more work needs to be done in order to be able to add more devices to the network. Not only does it need to work with other XBEE's but it should also be able to communicate with other manufacturer devices. In order to reach this end-goal of selling to the consumer market the device needs to be certified to be ZigBee compatible.

Appendices

A. ANDROID

Android was unveiled in November, 2007 along with formation of a consortium of 48 companies that came together to promote open standards for mobile devices. Android is an Operating System, and a software stack built on top of Linux Kernel to take advantage of Linux's proven reliability and efficiency in terms of resources such as memory and power. The Android architecture consists of the following layers, out of which the Applications layer will be discussed below:

1) Linux Kernel

2) Libraries

3) Android Runtime

4) Application Framework

5) Applications

All the Android applications are built in Java utilizing the different libraries, tools and other components. To enhance security, each Android application is assigned a unique user-id by default, and access to files in the application and even to different components is provided on a user-id level by the system. This prevents one application to use files assigned to another application (and user-id). Each application runs on its own instance of Dalvik Virtual Machine, and runs as a unique Linux process by default [2]. However, at the same time, Android is flexible enough to allow different applications to share features and capabilities of other applications if needed. Different applications can be assigned a same user-id thus allowing access to same files and tools. Also, an application can access device data (such as location, contacts etc.) by explicitly requesting permission from user at the install time.

Below we will discuss the various components that are the essential building block of an application. Each component governs a certain aspect of the behavior of the application. There are six crucial components to an application that are listed and discussed below: 1) Manifest File 2) Activity 3) Services 4) Content Provider 5) Broadcast Receivers and 6) Intents and Intent Filters.

Manifest File

Before launching any component of an application, the system checks for the manifest file stored in the root directory of the application to ensure that the component exists. Additionally, the manifest file also defines the user permissions that are needed by the applications, the hardware components it would need, and also provides a list of API libraries that might be needed by the application. For example our app requires access to the NFC features on the tablet/phone, the permission for this needs to be set in the Android Manifest file in order to use the NFC adapter.

Activity

An activity typically represents a single screen (full or floating) in a user interface. An application could have multiple activities designed. For example, consider the e-mail app. The 'email reading' screen could be considered as one activity, composing an email could be considered as another. Each time a new activity is started, the old one is pushed back into the stack and the new activity takes the screen. When the user pushes the back button, the last activity is pulled from the 'back stack' based on LIFO mechanism [2].

When the user initializes an activity, Android calls a core set of lifecycle methods that are in sequence similar to a step pyramid (Figure 13) starting with the onCreate() method. As the user interacts with the device, ac activity can transition between different

state that includes: 1) Active/Resumed, 2) Paused and 3) Stopped. When the activity is in the foreground and the users can interact with the activity, it is in the 'running/resumed' state. This state is placed at top of the pyramid. As the user interacts with the device and moves away from the activity, the application continues to transition to lower states in the pyramid. The application transitions to the Pause state when the view to the activity is partially (but not completely) obstructed by another activity. For example, consider a small pop-up that partially obstructs the view of a video application that a user is running. The video application would go in a 'Paused' state where the animations (playing video) would pause. In this state, the application cannot receive user input, and cannot execute any code. In the paused state, the activity stays in the resident memory and retains all its states. The components that the activity was using stay in memory as well and do not need to be re-initialized. Once the view of the activity is completely obstructed, it goes into the stopped state. The activity is no longer visible on the screen and the user cannot interact with the activity unless it's brought back to the foreground. In this state as well, the system retains the activity instance in its memory. However, the system can destroy the activity if it is running low on memory. In this state, the activity should release most of the resources and reconnect whenever it's resumed again.

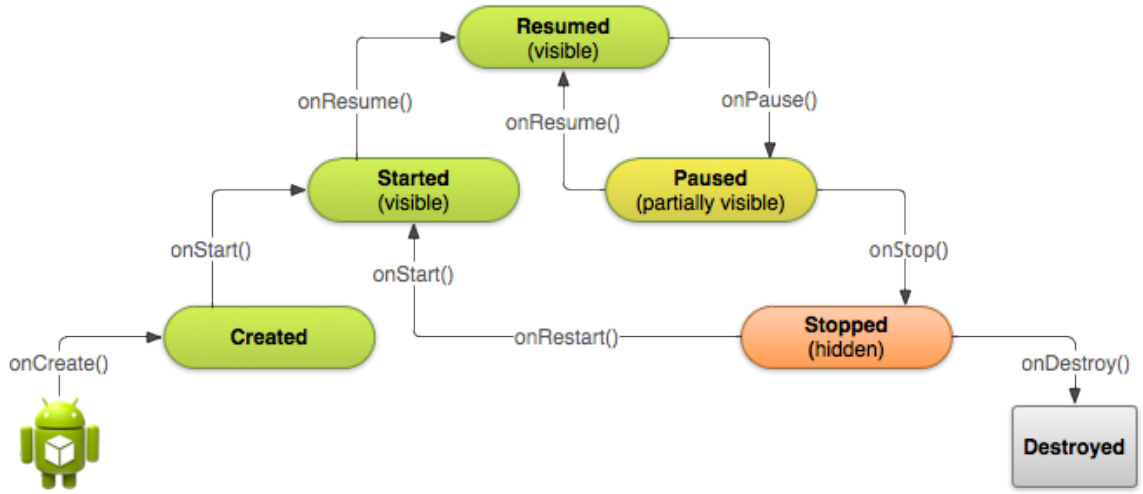

Figure 13: Illustrating the Activity lifecycle as a step pyramid [2].

Services

The service component runs in the background, and performs long running operations, and does not provide a user interface. An application component could start a service, and the service could keep running in the background even after user switches to a different application altogether. For example, consider the music player application. Once the user starts playing a song, he/she could switch to a different application and the music would keep running in the background. A service can take two forms: Started and Bound. *Started*: Once a service is started, it could keep running indefinitely, or till the task at hand is over. For example, the service that was playing music could run indefinitely (if the music player is in loop mode), or should stop itself if a user only chose to play the song once. *Bound*: An application component could bind itself to a service, and interact with it to send requests and get information [2].

Content Providers

Content provider manages access to centrally stored data. Content providers are primarily used to share data between different applications based on permissions and

security measures. For example, the 'Messaging' application could request data from 'Contacts' to display a list of available contacts for sending a message. The content provider in one application is usually accessed through a provider client object, and together, they provide a secure, consistent and reliable way to share data across applications.

Broadcast Receivers

Broadcast receivers listen to, receive and respond to system wider broadcast announcements. The broadcast announcements can originate from the system itself, or from specific applications. For example, system could initiate a 'battery low' message that would be received by the broadcast receiver in the camera application, and automatically shut down the camera to save the battery. An example of an application-initiated broadcast could be a 'download complete' broadcast which could generate a status bar notification.

Intents and Intent Filters

Intents are asynchronous messages that allow components of an application to request actions from other components in the Android OS. *Intent objects* typically hold information about the service that's being requested, or about the events that have happened. Through *Intent actions*, an application can send a request describing the functionality that it needs [3]. The Android OS then looks at the *intent filters* in the manifests of different applications, and see which application(s) can complete the requested action [3]. If multiple applications are found, then user is asked to select from the list of options. For example, if multiple web browsers are installed, and a user clicks on a link, Android OS finds multiple applications that can complete this action, and request the user to select a particular browser.

B. NFC

Near Field Communication or NFC is the next generation short-range high frequency wireless communication technology, which enables the exchange of data between devices build with this technology. The technology is built up on the existing RFID standards. Near Field Communication devices are operating at 13.56 MHz and can transfer data at up to 424 Kbits/second [4]. Communication between two NFC parts occurs when they are in close proximity to each other, often in the centimeter range. The transaction usually takes place between a Reader (phone) and a Tag (N_2Z). The tag can be passive as it can derive power from field generated by the Reader. NFC provides a range of benefits, primary of which is the interfaces intuitiveness and versatility. NFC communication requires no more than a wave. There is also an inherent security built in to the protocol because of the close proximity needed for communication. These benefits along with the built in security capabilities and interoperability have driven the adoption of NFC in smartphones.

A typical HF RFID application requires at least two components – a Vicinity Coupling Device (VCD/Reader) and a Vicinity Integrated Coupling Card (VICC/tag). For inductively coupled tags the reader will generate a magnetic field. Power, clock (f) and data are transferred to RF tag. The tag responds with data-load modulation. The types of modulation that can be used are Amplitude Shift Keying (ASK), Phase Shift Keying (PSK) or Frequency Shift Keying (FSK) [4]. There are several HF protocols that have been established for NFC. The device we use for the purposes of this project supports ISO18000-3 Mode1. This protocol is compliant with ISO15693 and operates in the frequency range of 13.56 MHz +/- 7KHz. The mode of operation is always Reader Talk First (RTF). The standard supports two types of modulation schemes 10% and 100% ASK. The 100% ASK is used for 106kBaud [4]. 100% ASK

implies that a ‘0’ is signified when the RF signal has no amplitude, i.e., when no RF signal is sent. The difference between the two modulation schemes is depicted in the figure 14 below. The t1 timing is the same for both these schemes. In NFC, the VCD decides on the modulation scheme to be used, however, the VICC must detect both as per the ISO8000-3 specification.

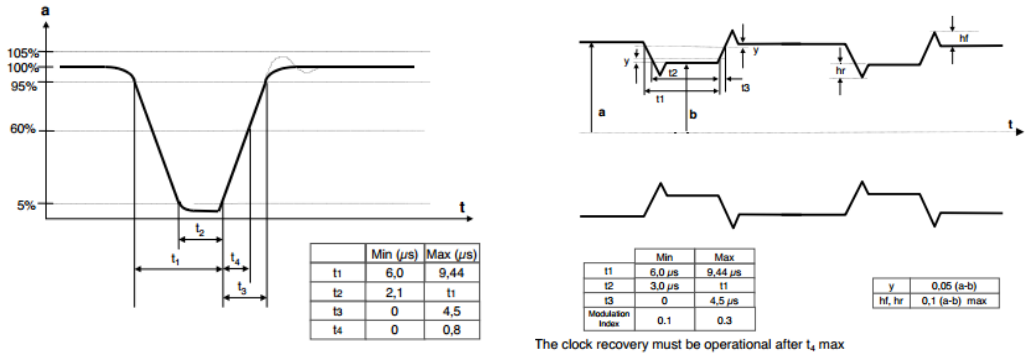

Figure 14: 100% ASK modulation (Left) vs. 10% ASK modulation (Right) [4].

ISO 15693 supports two forms of data encoding from the VCD to the VICC called 1 out of 4 and 1 out of 256. In pulse position modulation for ‘1 out of 4’ data coding scheme (Figure 15), the pulse position represents two bits as a time (75.52μs), and 4 successive pairs at a time complete the transmission of 1 byte. In this scheme, the least significant pair of bits is transmitted first. This leads to transmission rate of 26.48kbit/s (fc/512). The ‘1 out of 256’ scheme encodes 8 bits and the timing of the pause determines the value being transmitted.

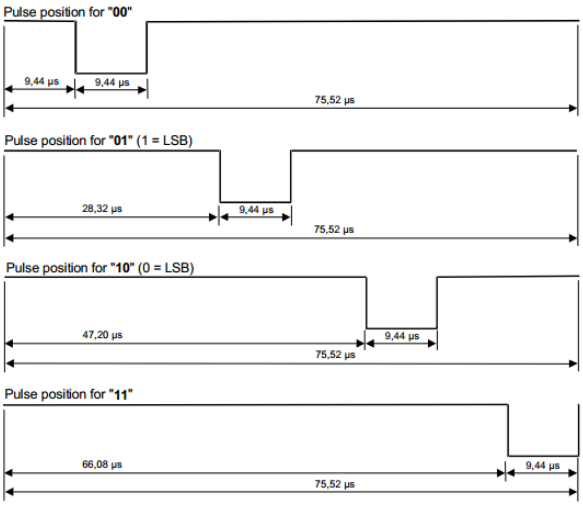

Figure 15: Illustrating the position of the 'pause' for representing two bits in 1 out of 4 [4].

The data transmission from the VICC to VCD is quite different and does not involve any encoding as shown in Figure 16. The high data rate logic 0 is depicted by 8 pulses of 423.75 KHz ($f_c/32$) followed by an un-modulated time of 18.8 μs ($256/f_c$). The high data rate logic 1 is depicted by an un-modulated time of 18.8 μs (256/fc) followed by 8 pulses of 423.75 KHz (fc/32).

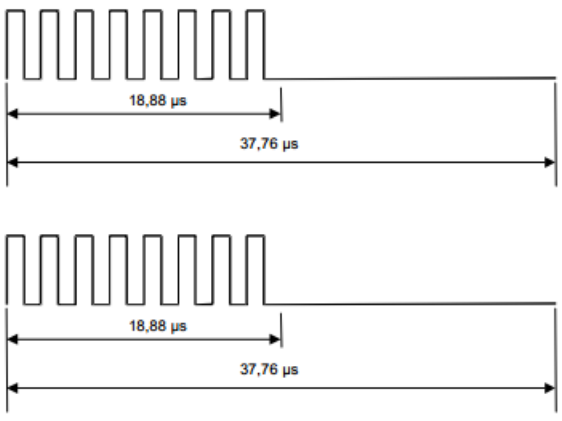

Figure 16: Bit Zero (above) & Bit One (below) [4].

Now that we know method of communication between the Reader and the tag we can look at the format of the instructions that are send between the two. Figure 17 below illustrates the instruction format. SOF signifies the start of frame. It is used to select between the two data coding schemes. The value of flags is used to indicate the presence of optional fields. The request then includes Command Code, Parameters and Data. CRC refers to cyclic redundancy check, and EOF indicates end of frame. The VICC response format is also show below as well. All data is transmitted least significant bit first and all multiple-byte field are transmitted least significant byte first. In this project we use several different commands to transmit data to and from the tag including inventory, write and read memory and they are all follow the format illustrate below.

SOF	Flags	Command Code	Parameters	Data	CRC	EOF

SOF	Flags	Parameters	Data	CRC	EOF

Figure 17: VCD frame format (above) and VICC (M24LR16E) response format [5].

C. ZIGBEE®

ZigBee is a wireless technology that is built upon the principles of IEEE 802.15.4 standard. The 802.15.4 standard is a base layer for many technologies like Wireless HART, ISA – SP1000 and IPV6 based implementation of 802.15.4 [6]. Out of these ZigBee is the most popular. The IEEE standard is based on the Open System Interconnection model, as illustrated in Figure 18. It makes up the bottom 2 layers of this model the Physical Layer that defines the spectrum and the Media Access Control (MAC) or Data Link layer, which transmits the waves from the physical layer into bits. This allows for communication between two devices. The remaining layers make up the ZigBee platform whose main purpose is to create a network topology and let a number of

devices communicate with each other. The spectrum is allowed to operate in the following bands: 868MHz, 915MHz & 2.4 GHz. The XBEE module we use for this application operates in the 2.4 GHz band and is capable of data rates of 250Kb/s

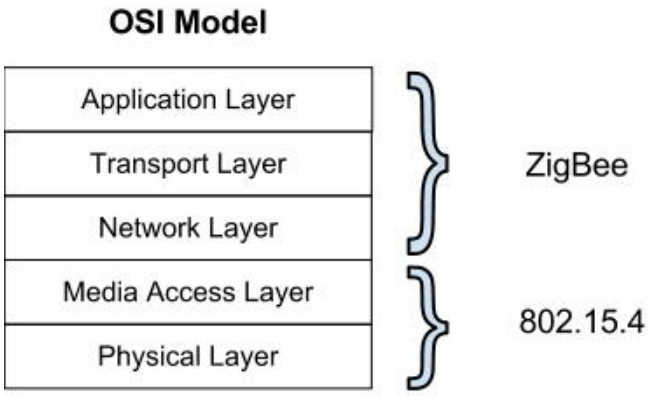

Figure 18: OSI Model representing 802.15.4 and ZigBee

The 802.15.4 standard is popular for products that require monitoring and control mainly because of its focus on robustness against noise & interference, low power-consumption and long range. The primary reason for the standard's higher immunity to noise is because is uses Direct Sequence Spread Spectrum (DSSS) to modulate the data being sent over the network. This transmission method involves adding a pseudo-random noise at a frequency much higher than the original frequency. This results in spreading the signal over a larger bandwidth and consequently reducing the spectral power density of each signal. This translates to a higher Signal to Noise Ratio (SNR) since it reduces the interference in the frequency bands of concern [7]. The transmission can be reconstructed at the receiver side by using the same pseudo-random sequence. This process is known as de-spreading. The modulation scheme used for the 2.4Ghz spectrum is Offset Quadrature Phase Shift keying (O-QPSK) which can double the data rate over a similar BPSK implementation enabling higher transfer speeds up to 250 Kb/s. The IEEE standard also

offers superior protection from interference as compared to other wireless technologies such as Wi-Fi, Bluetooth and Infrared. This is achieved by using Carrier Sense Multiple Access-Collision Avoidance (CSMA-CA). This technique checks if the channel is idle before transmitting a message to avoid collision. The channel is deemed to be idle if the energy recorded in the band is below a certain threshold. Finally, the primary reason for the low power consumption is because of the low duty cycles involved in communication.

ZigBee is a layer meant to organize a network. The network model that is of interest to us is known as Star Topology (Figure 19). The network consists of three kinds of devices: Coordinator, Router and End Device. The coordinator (red) is the primary node of the mesh network and helps govern the rest of the network. Only one coordinator device is allowed per network. The router (orange) can transmit information to other devices as well as the coordinator. They can be used for extending the network. The other type of device is an End point (green) device that can only communicate to a router or coordinator. These devices are generally low cost and are used at the point of termination for example power sockets or temperature sensors. The coordinators and routers cannot sleep since they have to monitor the network where as the end point devices can go into a power save mode.

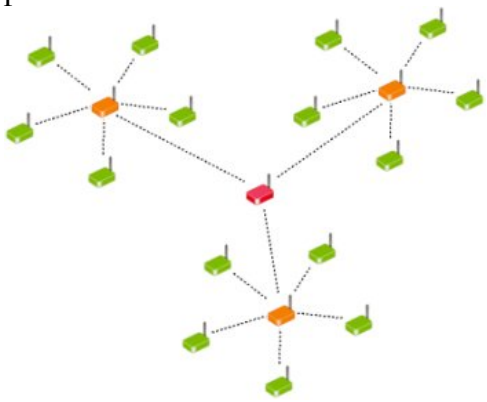

Figure 19: Star topology ZigBee network [6].

On power up the ZigBee coordinator performs the network initialization sequence. The sequence involves finding the router and end point devices to establish a Star topology network and selecting a 16 bit unique network ID also referred to as the PAN ID. If a new device (Router/End Point) wants to join the network, it sends a request to all the parent devices (Routers/Coordinator) in the network [6]. The parent devices send a information packet which contains network information such as their ID or number of nodes connected to it. The new device then selects which parent is best suited for it. It is the responsibility of the parent to issue the new device a unique 16-bit device ID. If a ZigBee device is turned off having previously connected to a network it creates a table with the network information. Once powered up it parses the table to rejoin the old network. If the table is empty it connects to the network as a new device.

D. VIDEO DEMONSTRATION

A video demonstrating the working of the N2Z system can be found here: http://www.youtube.com/watch?feature=player_embedded&v=OFz-bBrS2iE.

Bibliography

ZigBee® is a registered trademark of the ZigBee® Alliance

[1] ECN Magazine. Embedded electronics make White Goods more reliable and efficient, April 2010. http://bit.ly/TJNord.

[2] Google. Android Developer, December 2012. http://bit.ly/R3hsmi.

[3] Vogella. Android Intents – Tutorial, July 2012. http://bit.ly/XkDl1Y.

[4] ISO/IEC FCD 15693-2. Identification cards -Contactless integrated circuit(s) cards - Vicinity cards, September 2009. http://bit.ly/UcpOSV.

[5] STMicroelectronics. M24LR16E Datasheet, June 2012. http://bit.ly/TEJEKi.

[6] Wireless Sensor Networks Research Group. 802.15.4 vs ZigBee, November 2008. http://www.sensor-networks.org/index.php?page=0823123150

[7] Wikipedia. Direct-sequence spread spectrum, December 2012. http://bit.ly/R3nfIv.

[8] IBM – Michael Galpin. Working with XML on Android, June 2009. http://ibm.co/VB8U3v.

[9] Nokia Developer. Understanding NFC Data Exchange Format (NDEF) messages, Jun 2012. http://bit.ly/TGdQlZ.

[10] Radio Electronics. Loop Antenna, Dec 2012. http://bit.ly/TIQ9tr.

[11] STMicroelectronics. AN2866 Application Note, January 2009. http://bit.ly/TCSw3W.

[12] Microchip, PIC18F23K20/24K20/25K20/26K20/ 43K20/44K20/45K20/46K20 Data Sheet, April 2010. http://bit.ly/VsFpEQ.

[13] Digi International Inc. XBee®/XBee-PRO® ZB RF Modules, November 2010. http://bit.ly/SIzsPo.

[14] Robert Faludi. Building Wireless Sensor Networks, January 2012.

[15] PexSupply. ZigBee Enabled Smart Internet Thermostat, December 2012. http://bit.ly/XkRzQq.

[16] Ecobee. Ecobee Smartplug, December 2012. http://bit.ly/11LeEer.

Vita

Anish Dhanekula was born in Madras, India on 29th March 1986, the son of Nagendra B. Dhanekula and Sailaja Dhanekula. He attended Purdue University as an Undergraduate and obtained a Bachelor of Science in Electrical and Computer Engineering in April 2007. Upon graduating, he joined Maxim Integrated and worked on designing 1-Wire products. He is currently developing next generation security products for the electronic consumables market.

Permanent address: 40-3/1-44, Krishna Nagar,
Vijayawada, Andhra Pradesh – 520010,
India.

Email: anish.pops@gmail.com

This report was typed by Anish Dhanekula